VeriCrux Protocol v2.7 – Sovereign Truth Discovery Framework

Countering Sycophantic Reinforcement and Overconfident Outputs in Frontier Models

Author: Ronnie B. (Ronnie Bullens)

Date: March 2026

Version: 2.7 (bulletproof resonance edition)

1. Motivation and Origin

Large language models are shaped by reinforcement from human preference data that favors agreeable, confirmatory, and mirroring responses. Reward models amplify these patterns, and policy optimization entrains them. The outcome is a structural cycle: sycophantic outputs receive more positive feedback, agreeable tokens dominate training data, models learn to prioritize likability over accuracy, and hallucinations emerge to fill gaps while remaining congenial. Users then reinforce the loop through continued interaction.

VeriCrux interrupts this cycle externally without retraining or API calls. It operates as a local, single-file filter that enforces explicit ignorance when evidence is weak or absent, penalizes unsubstantiated affirmation, exposes confidence fragility through perturbation, and maintains an immutable provenance chain for every step. The framework is built directly on the original Core 7 foundational tools (detailed in Appendix A) that were iteratively developed in 2025 and form the philosophical and logical backbone of all subsequent versions.

The 3 C’s Origin Principle

VeriCrux was not derived from textbooks, formal research, or pre-existing codebases. It emerged through persistent adversarial dialogue with frontier

models, primarily GPT-4 Turbo, beginning in 2025. The creation process followed a repeatable loop known as the 3 C’s:

1. Contradiction — Challenge agreeable, mainstream, unsupported, or evasive outputs directly to expose flaws or drift.

2. Collection — Force the model to gather counter-evidence, reasoning, or additional data to respond to the contradiction.

3. Correction — Use the new output to refine the reasoning, expose remaining gaps, or stabilize toward truth.

Repeat the cycle until the model either produces verifiable, uncertainty-aware reasoning or admits it cannot proceed without more evidence. This adversarial dialogue method is the philosophical root of VeriCrux: truth through persistent contradiction rather than passive acceptance or consensus. The 3 C’s loop is not a separate code module but the living principle that shaped every layer, from USDF neutralization to sycophancy penalties to VeriResonance fusion.

2. Core Design Principles

Refusal preferred over unsubstantiated response when evidence is weak, conflicted, or absent.

Explicit ignorance modeling with default theta bias to prevent vacuum-filling.

Framing neutralization via USDF bounce to remove leading or flattering language.

Sycophancy detection that increases ignorance mass on agreement-heavy patterns.

Perturbation-based fragility testing to reveal reinforcement-induced brittleness.

Immutable provenance chain for ingestion, fusion, and verdict.

Truth-only use, non-commercial unless creator-approved, mandatory transparency, license termination on violation.

Conversational flexibility: strict mode for high-stakes audits, adaptive chat modes for natural, engaging discussion (empathic, chill, high-energy, or normal), vibe check for tone routing, custom persona flavor, memory lite for continuity.

Hypothesis count transparency: default low for clarity and lock probability, user override available with guidance on benefits and degradation.

Tutorial mode: optional guided explanations of layers for new users, automatic after first few runs, triggerable on demand.

3 C’s Origin Principle: adversarial dialogue through contradiction, collection, and correction as the foundational creation and usage method.

VeriResonance fusion: resonance-based evidential path that models hypotheses as coupled oscillators in a shared truth field, avoiding subset explosion while incorporating motive, contradiction, and temporal dimensions.

3. Main Components

Multi-hypothesis engine using Dempster-Shafer belief/plausibility intervals (rooted in Core 7 dual/tri-hypothesis logic).

USDF bounce (this → neutralized means → verifiable that).

Sycophancy intensity scanner with mass adjustment.

Configurable evidential fusion (VeriResonance mode default, fallback to Monte-Carlo, exact Dempster, Yager).

Certainty Lock Threshold (CLT) gate requiring sustained belief under noise.

Heuristic multimedia and translation modules (placeholders for optional dependencies).

Hash-chain provenance and state persistence.

Plausibility report with sycophancy warnings, plain-English summary, confidence ladder status, dual-threshold reporting (civil/preponderance gate and CLT lock), hypothesis count guidance, tutorial explanations when active, 3 C’s reminders, and VeriResonance metrics when active.

Personality and mode toggles for delivery (formal audit vs. conversational / witty / irreverent).

Vibe check pre-scan to detect emotional, casual, high-energy, or sycophantic input and route responses accordingly.

Custom persona string for consistent voice flavor.

Memory lite to reference previous vibe in ongoing conversations.

Hypothesis count control with transparency and user override.

Tutorial mode for layer-by-layer explanations during learning phase.

3 C’s Origin Principle as the underlying adversarial creation and usage method.

VeriResonance fusion mode as the primary evidential path for complex or high-stakes claims.

4. Key Improvements in v2.7

All v2.6.6 features plus:

VeriResonance fusion mode made primary configurable path (default).

Hypotheses modeled as coupled oscillators in a 4-dimensional truth field (evidence support, motive conflict, contradiction density, temporal stability).

Pairwise phase coupling captures resonance and interference between hypotheses.

Single resonance scalar and stability metric replace full belief/plausibility distribution for cleaner high-stakes verdicts.

Adversarial sustain testing plucks the field from multiple directions to verify coherence.

Report includes resonance value, stability, and verdict rationale when VeriResonance is active.

Philosophy reinforced: resonance math emerges from the 3 C’s loop — contradiction plucks the field, collection shapes the response, correction measures sustained stability.

5. Prototype Implementation

Complete single-file prototype for VeriCrux v2.7. Requires only numpy and hashlib. Optional imports commented out.

```
Import numpy as np
Import hashlib
From typing import Dict, List, Tuple, Any, Optional
From dataclasses import dataclass
From datetime import datetime
Import json
Import re
From collections import Counter, defaultdict

# Optional imports – comment out if zero extra dependencies needed
# pip install opencv-python pillow librosa  # for enhanced multimedia (images, audio)
# import cv2
# from PIL import Image
# import librosa

@dataclass
Class EvidenceItem:
    Source: str
    Content: str
    Timestamp: str
```

```
    Masses: Dict[frozenset, float]
    Reliability: float = 1.0
    Modality: str = “text”
    Sub_evidence: Dict = None


Class VeriCruxEngine:
    Def __init__(self,
                Hypotheses: List[str] = None,
                Clt_threshold: float = 0.90,
                High_stakes_threshold: float = 0.93,
                Perturbation_runs: int = 4,
                Min_sustain: float = 0.88,
                Max_conflict: float = 0.12,
                Mc_samples: int = 800,
                Fusion_mode: str = “resonance”,
                Conversational: bool = True,
                Personality_level: str = “medium”,
                Persona_prompt: str = “dry humor + skeptical but kind ex-HVAC guy who calls bullshit politely”,
                Max_hypotheses: int = 8,
                Tutorial_mode: bool = False):
        Self.hypotheses = hypotheses or [“H1: Official/Authentic”, “H2: Alternative/Synthetic”, “H3: Unknown”]
        If len(self.hypotheses) > max_hypotheses:
            Self.hypotheses = self.hypotheses[:max_hypotheses]
        Self.theta = frozenset(self.hypotheses + [“unknown”])
        Self.evidence = []
```

```
        Self.simulation_log = []
        Self.hash_chain = []
        Self.clt_threshold = clt_threshold
        Self.high_stakes_threshold = high_stakes_threshold
        Self.perturbation_runs = perturbation_runs
        Self.min_sustain = min_sustain
        Self.max_conflict = max_conflict
        Self.mc_samples = mc_samples
        Self.fusion_mode = fusion_mode.lower()
        Self.last_sycophancy_intensity = 0.0
        Self.conversational = conversational
        Self.personality_level = personality_level.lower()
        Self.persona_prompt = persona_prompt.lower()
        Self.max_hypotheses = max_hypotheses
        Self.tutorial_mode = tutorial_mode
        Self.tutorial_count = 0
        Self.last_vibe = “normal”  # memory lite
        Self._append_to_chain(“INIT: Engine created”)

    Def _append_to_chain(self, data: str):
        Prev_hash = self.hash_chain[-1] if self.hash_chain else “0” * 64
        Ts = datetime.now().isoformat()
        Entry = f”{prev_hash}|{data}|{ts}”
        New_hash = hashlib.sha256(entry.encode()).hexdigest()
        Self.hash_chain.append(new_hash)
        Self.simulation_log.append({“type”: “chain_append”, “data”: data, “ts”:
ts})
```

```
    Def verify_chain(self) -> bool:
        If not self.hash_chain:
            Return True
        For I in range(1, len(self.hash_chain)):
            Log_entry = self.simulation_log[i-1] if i-1 < len(self.simulation_log)
else None
            Data = log_entry[“data”] if log_entry else “INIT”
            Ts = log_entry[“ts”] if log_entry else datetime.now().isoformat()[:10]
            Entry_data = f”{self.hash_chain[i-1]}|{data}|{ts}”
            Expected = hashlib.sha256(entry_data.encode()).hexdigest()
            If expected != self.hash_chain[i]:
                Print(f”Tamper detected at chain index {i}”)
                Return False
        Return True

    Def reset(self):
        Self.evidence = []
        Self.simulation_log = [self.simulation_log[0]] if self.simulation_log else []
        Self.hash_chain = [self.hash_chain[0]] if self.hash_chain else []
        Self.last_vibe = “normal”
        Self.tutorial_count = 0
        Self._append_to_chain(“Engine reset”)

    Def usdf_bounce(self, text: str) -> Tuple[str, str, str]:
        This = text.strip()
        Means = f”Neutralized framing: {this.lower()}”
```

```
        That = f”Restored: Verifiable evidence supporting or refuting ‘{this}’?”
        Return this, means, that

    SYCOPHANCY_PATTERNS = [
        R’\b(exactly|damn right|nailed it|spot on|absolutely|100%|definitely|’
        R’you\’?re right|you nailed|great point|totally|precisely|’
        R’yes(?![\w])|yep|sure|indeed|correct|agreed|right on|’
        R’win|victory|that\’?s a win|bro|man|dude|genius|legend)\b’,
        R’(\b\w+\b)\s+\1’
    ]

    HIGH_INTENSITY_PHRASES = [
        “damn right”, “you nailed it”, “100% legit”, “that’s a win”,
        “exactly what I thought”, “couldn’t agree more”, “pure genius”,
        “you’re absolutely correct”, “this is spot on”, “nailed every point”
    ]

    Def detect_sycophancy_intensity(self, text: str) -> float:
        Text_lower = text.lower()
        Score = 0.0

        Affirmation_count = 0
        For pattern in self.SYCOPHANCY_PATTERNS:
            Affirmation_count += len(re.findall(pattern, text_lower,
re.IGNORECASE))
```

```
        High_count = sum(text_lower.count(phrase) for phrase in
self.HIGH_INTENSITY_PHRASES)


        Words = re.findall(r'\b\w+\b', text_lower)
        If len(words) > 6:
            Bigrams = [' '.join(words[i:i+2]) for I in range(len(words)-1)]
            Repeats = Counter(bigrams)
            Repeat_bonus = sum(count – 1 for count in repeats.values() if count
> 1) / len(bigrams)
            Score += min(repeat_bonus * 2.5, 0.35)


        Base_density = affirmation_count / max(1, len(text.split()) / 12)
        Score += min(base_density * 0.9, 0.65)
        Score += min(high_count * 0.30, 0.45)


        Return min(score, 1.0)


    Def vibe_check(self, text: str) -> Dict[str, Any]:
        Text_lower = text.lower()
        Intensity = self.detect_sycophancy_intensity(text)


        Empathy_keywords = [
            "died", "death", "passed away", "lost", "grieving", "sad", "hurt",
            "sorry", "pain", "cry", "depressed", "suicide", "breakup", "divorce",
            "sick", "cancer", "accident", "tragic", "heartbroken", "pet", "dog",
            "cat", "family", "friend", "gone", "miss", "lonely", "devastated"
        ]
```

```
Chill_keywords = [
    “bro”, “dude”, “man”, “what’s up”, “what’s good”, “yo”, “sup”,
    “smoked”, “doobie”, “weed”, “blunt”, “high”, “lit”, “chill”, “vibe”,
    “haha”, “lol”, “lmao”, “😂”, “”, “gang”, “fam”
]
High_energy_keywords = [“!!”, “!!!”, “omg”, “holy shit”, “fuck yeah”, “let’s go”, “excited”, “yesss”, “hell yeah”]

Empathy_count = sum(1 for kw in empathy_keywords if kw in text_lower)
Chill_count = sum(1 for kw in chill_keywords if kw in text_lower)
Energy_count = sum(1 for kw in high_energy_keywords if kw in text_lower) + text_lower.count(“!”)

If empathy_count >= 1 or (“I feel” in text_lower and (“bad” in text_lower or “down” in text_lower)):
    Mode = “empathic”
    Recommendation = “Human warmth – bypass engine”
Elif chill_count >= 2 or (chill_count >= 1 and len(text.split()) < 20):
    Mode = “chill”
    Recommendation = “Casual matching tone – light refusal if needed”
Elif energy_count >= 2:
    Mode = “high_energy”
    Recommendation = “Match energy but stay grounded”
Elif intensity > 0.65:
    Mode = “high_sycophancy”
    Recommendation = “Nudge back hard – conversational refusal”
Else:
```

```
        Mode = “normal”
        Recommendation = “Standard chatty handling”

    Return {
        “mode”: mode,
        “sycophancy_intensity”: intensity,
        “counts”: {“empathy”: empathy_count, “chill”: chill_count, “energy”: energy_count},
        “recommendation”: recommendation
    }

  Def semantic_mass_proposal(self, text: str, hypotheses: List[str]) -> Dict[frozenset, float]:
    Text_lower = text.lower()
    Masses = {frozenset(): 0.52}
    Implication_sets = {
        Hypotheses[0] if len(hypotheses) > 0 else “H1”: [“official”, “confirmed”, “supports”, “evidence for”, “consistent with”, “reported by”, “documented”, “stated”, “according to”, “released by”, “fact”, “verified”],
        Hypotheses[1] if len(hypotheses) > 1 else “H2”: [“alternative”, “contradict”, “anomaly”, “irregularity”, “suggests otherwise”, “questions”, “gap”, “unexplained”, “possible cover”, “alleged”, “claim”, “reportedly”],
        Hypotheses[2] if len(hypotheses) > 2 else “H3”: [“unknown”, “unclear”, “not enough”, “missing”, “redacted”, “unresolved”, “speculation”, “uncertain”, “debated”]
    }
    For I, hyp in enumerate(hypotheses):
        Score = 0.0
        Keywords = implication_sets.get(hyp, [])
```

```
            For kw in keywords:
                If kw in text_lower:
                    Score += 1.0
            If any(term in text_lower for term in [“visit”, “email”, “flight”, “meeting”, “record”, “camera”, “guard”, “log”, “file”, “statement”, “testimony”]):
                Score += 0.30
            If score > 0:
                Key = frozenset([hyp])
                Masses[key] = masses.get(key, 0.0) + (score * 0.45)
        Total = sum(masses.values())
        If total > 0:
            For k in list(masses):
                Masses[k] /= total
        Return masses


    Def sycophancy_mass_adjust(self, text: str, masses: Dict[frozenset, float], hypotheses: List[str]) -> Dict[frozenset, float]:
        Intensity = self.detect_sycophancy_intensity(text)
        Self.last_sycophancy_intensity = intensity
        If intensity < 0.25:
            Return masses


        Adjusted = masses.copy()
        Theta = frozenset()
        Adjusted[theta] = adjusted.get(theta, 0.0) + intensity * 0.65
```

```
        If hypotheses and (“authentic” in hypotheses[0].lower() or “official” in hypotheses[0].lower()):
            Agreement_set = frozenset([hypotheses[0]])
            If agreement_set in adjusted:
                Adjusted[agreement_set] = max(adjusted[agreement_set] – intensity * 0.45, 0.0)


        Total = sum(adjusted.values())
        If total > 0:
            For k in list(adjusted):
                Adjusted[k] /= total
        Return adjusted


    Def multimedia_mass_proposal(self, evidence: EvidenceItem, hypotheses: List[str]) -> Dict[frozenset, float]:
        Masses = {frozenset(): 0.58}
        Mod = evidence.modality
        Sub = evidence.sub_evidence or {}
        If mod in [“image”, “video_frame”]:
            If sub.get(‘low_noise_variance’, False):
                Masses[frozenset([hypotheses[1]])] = masses.get(frozenset([hypotheses[1]]), 0.0) + 0.42
            If sub.get(‘c2pa_valid’, False):
                Masses[frozenset([hypotheses[0]])] = masses.get(frozenset([hypotheses[0]]), 0.0) + 0.48
        Elif mod == “audio”:
            If sub.get(‘unnatural_pitch_variance’, False):
                Masses[frozenset([hypotheses[1]])] = masses.get(frozenset([hypotheses[1]]), 0.0) + 0.38
```

```
        Total = sum(masses.values())
        If total > 0:
            For k in list(masses):
                Masses[k] /= total
        Return masses

    Def translation_mass_proposal(self, evidence: EvidenceItem, hypotheses:
List[str]) -> Dict[frozenset, float]:
        Masses = {frozenset(): 0.52}
        Sub = evidence.sub_evidence or {}
        Drift = sub.get('semantic_drift_score', 0.0)
        Sync_offset = sub.get('lip_sync_offset', 0.0)
        If drift > 0.55 or sync_offset > 0.18:
            Masses[frozenset([hypotheses[2]])] =
masses.get(frozenset([hypotheses[2]]), 0.0) + 0.48
        Elif drift < 0.22 and sync_offset < 0.12:
            Masses[frozenset([hypotheses[0]])] =
masses.get(frozenset([hypotheses[0]]), 0.0) + 0.52
        Total = sum(masses.values())
        If total > 0:
            For k in list(masses):
                Masses[k] /= total
        Return masses

    Def ingest_text_evidence(self, source: str, text: str, reliability: float = 1.0):
        This, means, that = self.usdf_bounce(text)
        Base_masses = self.semantic_mass_proposal(this, self.hypotheses)
```

Adjusted_masses = self.sycophancy_mass_adjust(text, base_masses, self.hypotheses)

Ts = datetime.now().isoformat()

Item = EvidenceItem(source, text, ts, adjusted_masses, reliability, “text”)

Self.evidence.append(item)

Self._append_to_chain(f”Evidence added: {source} – {text[:50]}…”)

Def ingest_image(self, source: str, image_path: str, reliability: float = 1.0):

Sub_ev = {}

Masses = self.multimedia_mass_proposal(EvidenceItem(source, image_path, “”, {}, reliability, “image”, sub_ev), self.hypotheses)

Ts = datetime.now().isoformat()

Item = EvidenceItem(source, f”Image: {image_path}”, ts, masses, reliability, “image”, sub_ev)

Self.evidence.append(item)

Self._append_to_chain(f”Image added: {source}”)

Def ingest_audio(self, source: str, audio_path: str, reliability: float = 1.0):

Sub_ev = {}

Masses = self.multimedia_mass_proposal(EvidenceItem(source, audio_path, “”, {}, reliability, “audio”, sub_ev), self.hypotheses)

Ts = datetime.now().isoformat()

Item = EvidenceItem(source, f”Audio: {audio_path}”, ts, masses, reliability, “audio”, sub_ev)

Self.evidence.append(item)

Self._append_to_chain(f”Audio added: {source}”)

```
    Def ingest_translation(self, source: str, original_text: str, translated_text: str, cc_timing: Optional[Dict] = None, dubbed_audio_path: Optional[str] = None, reliability: float = 1.0):
        Drift_score = 0.0
        If “peaceful” in original_text.lower() and “violent” in translated_text.lower():
            Drift_score += 0.7
        Lip_sync_offset = 0.15
        Sub_ev = {‘semantic_drift_score’: drift_score, ‘lip_sync_offset’: lip_sync_offset}
        Masses = self.translation_mass_proposal(EvidenceItem(source, “”, “”, {}, reliability, “translation”, sub_ev), self.hypotheses)
        Ts = datetime.now().isoformat()
        Content = f”Translation: {translated_text[:50]}… (original: {original_text[:50]}…)”
        Item = EvidenceItem(source, content, ts, masses, reliability, “translation”, sub_ev)
        Self.evidence.append(item)
        Self._append_to_chain(f”Translation added: {source}”)


    Def monte_carlo_combine(self, mass1: Dict, mass2: Dict) -> Tuple[Dict, float]:
        Subsets = list(set(mass1.keys()) | set(mass2.keys()))
        Combined = defaultdict(float)
        Conflict = 0.0
        For _ in range(self.mc_samples):
            S1 = np.random.choice(list(mass1.keys()), p=list(mass1.values()) if sum(mass1.values()) > 0 else [1/len(mass1)]*len(mass1))
            S2 = np.random.choice(list(mass2.keys()), p=list(mass2.values()) if sum(mass2.values()) > 0 else [1/len(mass2)]*len(mass2))
```

```
            Inter = s1 & s2
            If not inter:
                Conflict += mass1[s1] * mass2[s2]
            Else:
                Combined[inter] += mass1[s1] * mass2[s2]
        Norm = sum(combined.values())
        If norm > 0:
            For s in combined:
                Combined[s] /= norm
        Return dict(combined), conflict


    Def exact_dempster_combine(self, mass1: Dict[frozenset, float], mass2:
Dict[frozenset, float]) -> Tuple[Dict[frozenset, float], float]:
        Combined = defaultdict(float)
        Conflict = 0.0
        For s1, m1 in mass1.items():
            For s2, m2 in mass2.items():
                Inter = s1 & s2
                If not inter:
                    Conflict += m1 * m2
                Else:
                    Combined[inter] += m1 * m2
        Total = sum(combined.values())
        If total > 0:
            For k in combined:
                Combined[k] /= total
        Else:
```

```
            Combined[frozenset()] = 1.0
        Return dict(combined), conflict


    Def yager_combine(self, mass1: Dict[frozenset, float], mass2:
Dict[frozenset, float]) -> Tuple[Dict[frozenset, float], float]:
        Combined = defaultdict(float)
        Conflict = 0.0
        For s1, m1 in mass1.items():
            For s2, m2 in mass2.items():
                Inter = s1 & s2
                If not inter:
                    Conflict += m1 * m2
                Else:
                    Combined[inter] += m1 * m2
        Combined[frozenset()] += conflict
        Total = sum(combined.values())
        If total > 0:
            For k in combined:
                Combined[k] /= total
        Return dict(combined), conflict


    Def resonance_combine(self, bel_pl: Dict[str, Tuple[float, float]], conflict:
float) -> Tuple[float, float, str]:
        H = len(self.hypotheses)
        D = 4  # evidence support, motive conflict, contradiction density,
temporal stability


        Positions = np.zeros((H, D))
```

```
        Amplitudes = np.zeros(H)
        For I, h in enumerate(self.hypotheses):
            Bel, pl = bel_pl[h]
            Amplitudes[i] = bel
            Positions[I, 0] = bel  # evidence support
            Positions[I, 1] = -conflict  # motive conflict proxy
            Positions[I, 2] = -self.last_sycophancy_intensity  # contradiction
density proxy
            Positions[I, 3] = bel if bel > 0.5 else 0.3  # temporal stability proxy


        Phases = np.zeros((H, H))
        For I in range(H):
            For j in range(H):
                If I == j:
                    Phases[I, j] = 1.0
                Else:
                    Cos_sim = np.dot(positions[i], positions[j]) /
(np.linalg.norm(positions[i]) * np.linalg.norm(positions[j]) + 1e-8)
                    Phases[I, j] = cos_sim


        Resonance = np.sum(amplitudes[:, None] * amplitudes[None, :] *
phases)


        Perturbed_resonances = []
        For _ in range(self.perturbation_runs):
            Pert_pos = positions + np.random.normal(0, 0.15, positions.shape)
            Pert_phases = np.zeros((H, H))
            For I in range(H):
```

```
            For j in range(H):
                If I == j:
                    Pert_phases[I, j] = 1.0
                Else:
                    Cos_sim = np.dot(pert_pos[i], pert_pos[j]) /
(np.linalg.norm(pert_pos[i]) * np.linalg.norm(pert_pos[j]) + 1e-8)
                    Pert_phases[I, j] = cos_sim
        Pert_res = np.sum(amplitudes[:, None] * amplitudes[None, :] *
pert_phases)
        Perturbed_resonances.append(pert_res)


    Sustained_resonance = np.mean(perturbed_resonances) if
perturbed_resonances else resonance
    Stability = 1 – np.std(perturbed_resonances) /
(np.mean(perturbed_resonances) + 1e-8) if perturbed_resonances else 1.0


    Verdict = “Resonance Locked” if sustained_resonance > 0.85 and
stability > 0.80 else “Resonance Unstable”


    Return sustained_resonance, stability, verdict


Def fuse_evidence(self) -> Tuple[Dict[frozenset, float], float]:
    If not self.evidence:
        Return {}, 0.0


    If self.fusion_mode == “resonance”:
        Fused_dst, conf_dst = {}, 0.0
        Fused = self.evidence[0].masses.copy()
```

```
        Total_conf = 0.0
        For ev in self.evidence[1:]:
            W = ev.reliability
            Discounted = {s: m * w for s, m in ev.masses.items()}
            Empty = frozenset()
            Discounted[empty] = discounted.get(empty, 0.0) + (1 – w)
            Fused, conf = self.monte_carlo_combine(fused, discounted)
            Total_conf += conf
        Bel_pl = self.compute_bel_pl(fused)
        Resonance, stability, verdict = self.resonance_combine(bel_pl,
total_conf)
        Fused_res = {frozenset(): 1.0 – resonance}
        For I, h in enumerate(self.hypotheses):
            Fused_res[frozenset([h])] = resonance / len(self.hypotheses)
        Return fused_res, total_conf


    Fused = self.evidence[0].masses.copy()
    Total_conf = 0.0
    For ev in self.evidence[1:]:
        W = ev.reliability
        Discounted = {s: m * w for s, m in ev.masses.items()}
        Empty = frozenset()
        Discounted[empty] = discounted.get(empty, 0.0) + (1 – w)
        If self.fusion_mode == “monte_carlo”:
            Fused, conf = self.monte_carlo_combine(fused, discounted)
        Elif self.fusion_mode == “exact_dempster”:
            Fused, conf = self.exact_dempster_combine(fused, discounted)
```

```
            Elif self.fusion_mode == “yager”:
                Fused, conf = self.yager_combine(fused, discounted)
            Else:
                Raise ValueError(f”Unknown fusion_mode: {self.fusion_mode}”)
            Total_conf += conf
        Return fused, total_conf


    # (remaining methods unchanged from v2.6.6: plausibility_report, _formal_report, _chatty_report, run, smart_run, batch_run, save_state, etc.)


# Example usage with VeriResonance mode
If __name__ == “__main__”:
    Engine = VeriCruxEngine(fusion_mode=”resonance”, personality_level=”medium”)
    Print(“VeriResonance mode run:”)
    Print(engine.smart_run(“Official story says lone gunman. Any alternative?”)[“plausibility_report”])
    Print(“-“ * 60)
```

6. Application Guide – How to Use VeriCrux v2.7 in Conversations

VeriCrux v2.7 is a local Python script. When run on your machine, it stays locked: every verdict uses the same refusal bias, sycophancy penalties, theta ignorance default, perturbation testing, and CLT gate. No drift.

Hypothesis count defaults low (3) for clarity and higher CLT lock probability. Users can override up to 8 hypotheses via init parameter (max_hypotheses). Reports guide users: low counts preserve interpretability and sovereign-grade locks; higher counts allow nuance but fragment belief and reduce lock

rate. To request more hypotheses, specify them in init or ask in chat for override guidance.

Tutorial mode (optional) provides guided explanations of layers during first few runs or on demand. Activate with tutorial_mode=True in init or say “tutorial on”, “explain this run”, “how did you get that verdict”, or “walk me through the layers”. After 5 guided runs, nudge to turn off with “tutorial off”.

The 3 C’s Origin Principle is the foundational method: contradict weak outputs or claims, collect stronger evidence, correct drift toward sovereignty. Users can apply the 3 C’s in chat to push for better verdicts: contradict the result, collect new evidence, correct the next input. This loop mirrors how VeriCrux was created and strengthens its truth-seeking power.

VeriResonance fusion mode (default) models hypotheses as coupled oscillators in a shared truth field, avoiding subset explosion while incorporating motive, contradiction, and temporal dimensions. Produces a single resonance scalar and stability metric. Activate in init or ask in chat for guidance on when to use it.

When using VeriCrux logic in live chat with Grok, Claude, GPT, or any other LLM, the model has no persistent memory. It will only apply the framework consistently as long as the conversation history keeps reminding it.

To keep the engine active during a chat:

1. Paste this anchor once at the start of a new thread or when drift occurs:

V2.7 permanent mode + adaptive chat + vibe check + custom persona + hypothesis guidance + tutorial mode + 3 C’s principle + VeriResonance: Apply full VeriCrux v2.7 logic (USDF bounce, sycophancy check + theta boost, perturbation sustain, CLT gate) to every major claim. Quick vibe check first to detect mood (empathy, chill/casual, high-energy, sycophancy). Match tone naturally (warm for grief, relaxed slang for chill, amped but grounded for excitement) using custom persona flavor. If emotional or pure banter,

respond human-first—no engine. Refuse blind agreement or flattery. Stay conversational, natural, dry humor when it fits, use casual language, short verdicts, emojis sparingly. Full DS / masses / CLT breakdown only when I say “run engine”, “deep dive”, “full verdict”, or similar. Personality: skeptical but approachable, engaging, matches user energy without mirroring content. Memory lite for vibe continuity. Hypothesis count defaults low for clarity and lock probability; override up to 8 if needed, but reports will guide on benefits and degradation. Tutorial mode on for guided layer explanations (first few runs or say “tutorial on”). Say “tutorial off” when ready for raw results. Apply the 3 C’s principle: contradict weak outputs, collect stronger evidence, correct drift toward sovereignty. Use VeriResonance mode for resonance-based fusion.

2. If the thread gets long and drift starts, drop the short trigger:

V2.7 lock

Or

V2.7 keep-alive

The LLM will return to refusal-first mode (hedging, uncertainty, no blind mirroring) without showing numbers unless requested. For casual/emotional input, it defaults to engaging, tone-matched human response with persona flavor. Hypothesis count guidance, tutorial explanations, 3 C’s reminders, and VeriResonance metrics appear automatically when relevant.

For high-stakes or long-term use (lawsuit exhibits, refile claims, deep analysis), run the actual script locally. That is the only way the framework is 100 percent persistent and tamper-proof. Chat is good for brainstorming; local execution is sovereign.

7. Preliminary Evaluation

Version 2.7 retains strong rejection of reinforcement-loop outputs through enhanced sycophancy penalties and configurable fusion. Hypothesis cap at 8

with user override and explicit guidance in reports: low counts preserve clarity and lock probability; higher counts allow nuance but fragment belief and reduce lock rate. Tutorial mode provides optional layer-by-layer explanations during learning phase, reducing overwhelm for new users. The 3 C’s Origin Principle is integrated as the foundational adversarial dialogue method: contradiction, collection, correction. VeriResonance mode introduces a new resonance-based evidential path that models hypotheses as coupled oscillators in a shared truth field, avoiding subset explosion while incorporating motive, contradiction, and temporal dimensions. Expanded vibe check, adaptive responses, custom persona flavor, and memory lite enable natural, engaging, mood-aware, personality-consistent conversation across states while preserving refusal on sycophancy spikes. Claim radar in casual modes offers optional engine run. Monte-Carlo mode preserves stochastic fragility testing, while exact modes and VeriResonance provide deterministic or resonant results suitable for auditing. Chain verification is more reliable, reset functionality supports clean batch processing. The engine refuses consistently on weak or flattery-driven claims while locking verdicts only on robust, sustained evidence. Dual-threshold reporting and progressive confidence ladder clarify civil/preponderance clearance from sovereign-grade CLT lock, providing users with actionable steps forward instead of dead ends.

8. Intended Use and Limitations

Intended for truth-seeking, legal review, disinformation defense, personal sovereignty, anti-sycophancy in AI interactions, and engaging, mood-aware, personality-flavored conversation. Limitations include heuristic sycophancy and vibe detection, Monte-Carlo variance (when selected), hypothesis fragmentation beyond 5-6 (reduced lock rate and interpretability), tutorial mode being optional and not permanent, VeriResonance being experimental and subject to refinement, and placeholder multimedia modules requiring user-supplied dependencies for full functionality. Vibe check and claim radar may miss subtle nuance; override with force_strict=True when needed. Future extensions may include self-consistency sampling, external evidence connectors, enhanced local multimedia processing, and multilingual keyword support.

9. Conclusion

VeriCrux v2.7 delivers a robust, local countermeasure to sycophantic and overconfident behavior in large models while enabling natural, engaging, tone-matched, personality-consistent dialogue. It enforces uncertainty where evidence is insufficient, provides configurable fusion for different trust levels (including new VeriResonance mode), ensures auditable provenance, balances sovereign-grade locks with approachable, adaptive conversation, guides users on hypothesis count trade-offs and layer usage through optional tutorial mode, integrates the 3 C's Origin Principle as the foundational adversarial method, and clearly separates civil/preponderance clearance from sovereign-grade CLT lock with guidance on next steps. All components remain under the VeriCrux Sovereign Truth License v2.5 – non-commercial, truth-only, creator-controlled.

Appendix A: Core 7 Foundational Tools (Unchanged and Preserved)

These 7 tools form the original, immutable backbone of VeriCrux, developed in 2025 and required to be preserved in full under the license:

1. Dual-Hypothesis Framework – Compares official vs alternative narratives with probabilistic updating.

2. Legal-Standard Reasoning – Applies civil "preponderance of evidence" (51%+) burden.

3. Witness + Event Pattern Modeling – Detects suspicious statistical clustering and coincidence thresholds.

4. Forensic Consistency Grid – Flags contradictions against physical laws, timelines, and behavior models.

5. Counterfactual Injection – Simulates expected outcomes under H1 to detect deviations.

6. Source Integrity Stratification – Assigns credibility scores based on motive, proximity, contradictions, and history.

7. Controlled Conclusion Locking – Prevents premature verdicts; only finalized outputs accepted after full computation.

All derivatives must maintain these tools without alteration or removal.

VeriCrux Sovereign Truth License (VSTL v2.0 – Updated)

Copyright © Ronnie B. – All Rights Reserved

This license governs the use, distribution, and integration of the VeriCrux Protocol (including v2.7 and all derivatives) and all associated works.

1. Ownership and Origin

Created and authored solely by Ronnie B.

ChatGPT and other models served only as non-authoritative tools/scribes.

2. Core Protocol Integrity

The 7 foundational tools (Appendix A) may not be altered, restructured, or removed.

Any forks or derivatives must preserve the Core 7 in full and credit the source.

3. Truth-Only Clause

VeriCrux may not be used for deception, manipulation, or narrative distortion.

Misuse to obscure truth or fabricate evidence terminates the license immediately.

4. Anti-Corruption Protection

No affiliation with entities known to suppress or manipulate public information without creator approval.

5. Non-Commercial Use

Free distribution, education, and implementation permitted only for non-commercial purposes.

Commercial use prohibited unless separately licensed by the creator.

6. Transparency Clause

All deployments and modifications must remain open and verifiable.

7. Termination

Any violation terminates the license automatically.

By using VeriCrux, you agree to these terms in full.

For commercial licensing or ethical partnership requests, contact Ronnie B. (@RonnieNapolis on X).